# An overview of CONLAN: a formal construction method for hardware description languages

R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, D. Hill, P. Skelly

INFORMATION PROCESSING 80, S.H. Lavington (ed.)
North-Holland Publishing Company


# AN OVERVIEW OF CONLAN: A FORMAL CONSTRUCTION METHOD FOR HARDWARE DESCRIPTION LANGUAGES

Robert PILOTY
Technische Hochschule Darmstadt

Dominique BORRIONE
Université de Grenoble

Frederick HILL
University of Arizona, Tucson

Mario BARBACCI
Carnegie-Mellon University

Donald DIETMEYER
University of Wisconsin, Madison

Patrick SKELLY
Honeywell, Phoenix

In this paper the motivation, objectives and basic concepts of the CONLAN family and its primitive set members are described. To promote an orderly development of hardware description languages and to enhance their acceptance in an industrial environment, CONLAN provides a powerful construction mechanism for such languages based on a common core syntax. Semantically related languages can be constructed which permit the description of digital systems at different levels of abstraction. Capabilities for syntax modification permit the suppression of unneeded constructs and the introduction of shorthand for frequently used objects to obtain simple yet useful languages.

## 1. INTRODUCTION

The decision to start the CONLAN project was motivated by the following assessment of the situation in the area of HDL and of Computer Aided Design (CAD) tools based on them:

Several dozen HDL's existed in 1973 [4] and every year since new languages have been proposed and published mostly from persons in academic institutions [5,6,7]. This tendency to proliferation is in sharp contrast to acceptance in industry. Neither have they been used to document the design process of digital systems nor to support tools for certification, synthesis and performance evaluation to any appreciable extent. Most CAD-tools in industry are designed to aid the manufacturing process (placement, routing, mask layout etc.). The process of systems and logic design is mostly carried out in the traditional way of drawing block and circuit diagrams at the IC package or gate level. In many cases these diagrams are the only true and complete documentation of the system. Most other aspects or phases of the system design, particularly system behaviour, are informally and incompletely described. Simulation as a means for advanced certification is used, if at all, at a very low level (mostly gate level) and hence at enormous cost for more complex systems. Most of the certification is done very late at the level of a physical prototype causing costly changes in physical design.

This situation has not changed very much in the four years of CONLAN development: HDL's continue to proliferate [8,9] but their usage in real life design has not increased in the same proportion. Only recently a growing interest in efficient tools for design support at systems and logic level can be observed, probably due to the advance of LSI and VLSI, where late changes make a system more and more costly, and due to increased system complexity in a competive market, calling for more efficient design tools [10,11].

There are several reasons why acceptance of existing HDL's is so low:

1. None of the languages alone is of sufficient scope to portray all aspects of a system and cover all phases of the design process.
2. Languages of different scope are syntactically and semantically unrelated.
3. Few of the languages are formally defined.
4. Only a few languages are implemented.
5. Descriptions are represented by character strings rather than diagrams.
6. There exists no comprehensive hardware and firmware design methodology telling how to use HDL's effectively.

The main aim of the CONLAN Working Group is to remedy the first four deficiencies [12]. Its primary objectives are:

1. To provide a common formal syntactic and semantic base for all levels and aspects of hardware and firmware description, in particular for descriptions of system structure and behaviour.
2. To provide a means for the derivation of user languages from this common base
   - having a limited scope adjusted to a particular class of design tasks,
   - thus being easy to learn and simple to handle,
   - yet having a well defined semantic relation among each other.
3. to support CAD-tools for documentation, certification, design space exploration, synthesis and so on.

CONLAN is not intended as a language standard trying to impose a certain style of hardware description on makers of design tools. It should rather be viewed as a formal system which allows them to construct HDL's of their choice in a consistent and unambiguous way within some notational conventions.

Character strings using the ISO-IRV 646 character set are the basic means of representing CONLAN descriptions, since they are more general and easier to use as an input or output for CAD-tools. However it is an objective of the CONLAN Group at a later stage, to show how to write descriptions in CONLAN which are isomorphic to space structure (network descriptions) or behaviour (sequentiall and concurrent flow descriptions) and to propose graphical members of the CONLAN family.

Problems of design methodology are not treated in this presentation of CONLAN. Much remains to be done in this area although a number of significant results have been obtained already.

## 2. BASIC APPROACH

CONLAN supports a self-defining extensible family of languages. Its member languages are tied together by a common core syntax and a common semantic definition system. The CONLAN construct to define a member language is called a language definition segment.

The member languages are used to write descriptions of hardware, firmware or software modules. Description definition segments are provided for this purpose as a CONLAN construct.

The method for semantic definition is based on the concept of abstract data types, which has been developed for programming languages like CLU [13] or ALPHARD [14]. An abstract data type henceforth called a type, is defined by a domain of elements and a set of operations on these elements. New types may be defined in terms of primitive types supplied with the language.

Contrasting the application of types in programming languages, the primitive types of a CONLAN member language except Primitive Set CONLAN (the lowest language level in the CONLAN family) are not implied. Rather they are defined in terms of the types of one other member language. This language is called the reference language of the language being defined. This establishes a partial order among the member languages: A language Lb is derived from a language La if La is the reference language of Lb or if there is a chain of reference languages leading from Lb to La.

In CONLAN the same construction mechanism and the same notational system used to provide descriptions is also used to define new language members. In this sense CONLAN is self-defining in contrast to externally defined languages using a separate language to define its semantics e.g. the formal description of PL/I using the Vienna Definition Language [15].

The CONLAN family of languages is open ended. New languages may be derived from existing ones at any given point in time as the need arises. They in turn may be used later as reference languages for further languages.

The syntax of a new CONLAN member may be made to differ from the syntax of its reference language by adding and/or deleting productions in the reference syntax. This capability permits the language designer to keep the syntax of a new language as simple as possible and yet allows the incorporation of new constructs to denote specific features e.g. the introduction of an infix symbol to denote some new operation. There is a set of productions which may never be deleted, and thus is common to all CONLAN members. It is called the core syntax.

The CONLAN text structure is shown in Fig. 1. At any given point in time it consists of a set of language definition segments and a set of description segments. Each segment is under the scope of a REFLAN statement. In a language definition segment this statement points to the language from which the new language is directly derived. Thus language LL1 is directly derived from L1 and LL2 is directly derived from L2. In a description segment the REFLAN statement points to the language in which the description is written.

There is one root language called base conlan (bcl), serving as the interface between the CONLAN Working Group and its public of toolmakers and users. Toolmakers start from bcl to construct languages and their associated CAD tools. Users write descriptions of hardware and firmware systems in these languages. Bcl provides a carefully chosen set of basic types reflecting the CONLAN concept of time and space, of signals and carriers, of arrays and records. It is described in more detail in [2].

To define bcl the CONLAN Working Group used a very low level but powerful language called "primitive set conlan" (pscl) to formally define the concepts represented by the bcl types. Pscl has no reference language. It owns a set of primitive types whose domains and operations are introduced informally. Pscl is described in more detail in [1].

It is important to note the CONLAN concept of hiding types and operations. Refering to Fig. 1 for the writer of a LL1 only those types and operations which are defined in L1 and not marked PRIVATE are visible and accessible. This implies that the types and operations of bcl are inaccessible to LL1, unless explicitly brought forward by L1 with a CARRY statement. The CARRY statement avoids the need for redefinition if a type is used in more than one language level. This hiding mechanism is the main instrument to keep derived languages simple and maintain a clear semantic relation with their ancestor languages.

## 3. FORMAL CONCEPTS

CONLAN operations are defined in operation definition segments and are normally denoted by an identifier prefixing a list of parameters. Operator symbols may be introduced for prefix and infix notation either in lieu of a standard functional notation or as an alternative to it via syntax modification.

Two categories of operations are known in CONLAN: activities and functions. An activity changes the status of one or more carriers. A function maps a domain of elements into a range of elements.

Types are defined in type definition segments of the following form (simplified):

```
TYPE t2(typed-parameters)BODY
set-definition
carry
operation-definition
ENDt2
```

Parameters are optional. If present, a family of types is specified by the definition. The set definition part serves to specify the domain of the type. In the operation definition part, operations may be defined to operate on the new type.

A description definition segment is used to define the input/output relation of hardware, firmware, or software modules. Details about their definition and usage can be found in [3].

```
DECRIPTIONnand(INx,y:btm1,OUTz:btm1)BODY
z.=(x∼&y)△1
ENDnand
```

In the above example △ is the delay operator, .= denotes terminal connection, and btm1 has been defined as the type "boolean terminal with default value 1" in [2].

A CONLAN member language is defined via a language definition segment of the form:

REFLAN oldlang CONLAN newlang BODY...ENDnewlang

The constituents of the body are:

- list of carried types and operations from the reference language (optional)
- definition of public and private types
- operation definitions (optional)
- description definitions (optional)
- format statements (optional)

Operations may be defined outside type segments. A library of operations could therefore be defined as part of a CONLAN segment. The same applies to descriptions if the language is geared to describing systems in terms of standard modules (e.g. TTL-modules). Syntax modifications via format statements are illustrated in [2].

All types referenced in the formal parameter lists appearing in non-private type, operation and description definitions must either be explicitly defined or carried from the reference language (closure of a language with respect to types).

## 4. CONLAN ENVIRONMENT

A CONLAN document has significance only if it is read by a person or machine. That reader (environment) is required to use available facilities to respond to and interact with the document. It must provide the type checking mechanism. It must record the names of defined and declared items and provide the data base they require. It must record signal values. From such records it can determine facts of importance to continued document evaluation. "System interfaces" are prescribed environment responses, not formally defined via CONLAN syntax.

Three broad classes of objects are of primary concern in working members of the CONLAN family:

"Values" are static objects; they do not change with time. An integer, a character, etc. are values.

"Signals" are lists of values. A different time is associated with each value. A signal is then a history of values.

"Carriers" are containers for values or signals. These values or signals can be replaced as a result of an operation invocation.

### 4.1. CONLAN Model of Time

CONLAN provides a discrete model of continuous, real time.

Real time is broken into uniform durations called "intervals" identified with integers greater than zero. Ascending, successive integers are associated with contiguous intervals. No relation between the interval and the real time second exists in general. An implementation may impose such a relation or permit users to specify such a relation.

At the beginning of each interval there are an indefinite number of computation "steps" identified with integers greater than zero. Successive steps provide a before/after relation only.

Values obtained at the last step of computation are the values associated with the interval.

When modeling a specific digital system satisfactory results are obtained at reasonable computational cost by quantizing time to some fraction of the second; for purposes of example assume the nanosecond. Actual signals are then constrained by this quantization to change at the boundaries of 1 ns. durations. Computing the value of a specific signal during a specific 1 ns. duration may require successive computations: if a wire is driven by a gate network modelled by a&b | c&d, then a&b and c&d must be evaluated before the signal value is determined. The CONLAN interval and step support this model of digital hardware and method of simulation (Figure 2).

No real time is thought to elapse when evaluating a mathematical function or executing a computer program. Yet many successive computational steps are usually required. Again the CONLAN model of time supports such computation.

### 4.2. CONLAN Model of Computation

In order to model real hardware components, some mechanism to describe delays in components and wires must be provided. The solution adopted in CONLAN is to keep the history of values computed at every step of every interval. Separate histories (called "signals" in CONLAN) are kept for each component, pin, wire, etc. of the hardware system. Signals are abstractions and do not have a physical interpretation. To provide the link between the signal (i.e. a history of values) and the component, a special type of object, called a "signal-carrier" is provided by the language.

Hardware descriptions record how the signal parts of some carriers are related to those of other carriers. These relations display behaviour and/or organization and support computation of unknown signal parts. Such computation is usually performed viewing past and present signal values as "known" and future values as "unknown". With each computational step, known values are used to determine a future value and thereby change its status to known.

The interval and step counters are managed by the environment. The contents of these counters are made available to toolmakers via t◌̃ and s◌̃.

t◌̃ is an integer whose value is the current time interval. Contiguous values are provided in ascending order starting with one. s◌̃ is an integer whose value is the current computation step. Contiguous values are provided in ascending order, starting with one.

When the environment determines that all signals have attained stable values, it increments the value provided by t◌̃ and resets the s◌̃ counter to 1. It detects computation step oscillation (s◌̃ reaches a predetermined limit) and responds to it with a message and optionally

termination of document evaluation or continuation using the signal values available at the last step of computation.

Base CONLAN defines several data types to support the models of time and computation sketched above:

A Computational Step Signal (cs-signal) is the mechanism used to record a history of values during one real time interval. The values to be recorded must all be of the same type, and the type must be specified when a cs-signal is declared. Thus, one could have cs-signals recording values of type integer, boolean, etc. For instance the set of cs-signals carrying Boolean values is:

```
{
(.0.),(.1.),          "/cs-signal of length 1/"
(.0,0),(.0,1.),(.1,0.),(.1,1.)
                      "/cs-signal of length 2/"
(.0,0,0.),(.0,0,1.),(.0,1,0.),...
                      "/cs-signal of length 3/"
.....
}
```

A Real Time Signal (signal, for short) is the mechanism for recording interval values. During an interval, an unlimited number of computation steps may occur and these are recorded in cs-signals. A signal consists of a tuple of these cs-signals.

For instance, assume a Boolean signal (s), with the following history:

(.(.0,0),(.0,1.),(.1,1,0.).)

Pictorially this is represented as:

| Real Time Interval | Computation Step | Value |
|---|---|---|
| 1 | 1 | 0 |
| | 2 | 0 last |
| 2 | 1 | 0 |
| | 2 | 1 last |
| 3 | 1 | 1 |
| | 2 | 1 |
| | 3 | 0 last |


Signal Carriers (carriers for short) are derived from primitive type cell∅ (defined in pscl [1]). Each cell contains a signal. Carriers are declared by specifying the type of values recorded by the signal (x) and a default/initial part (di). The role of the default/initial value is to provide a value to be used in some operations, as described below.

Terminals are carriers with no retention properties. If no connect activity is invoked during a computation step to extend the signal component of a terminal, then the system extends that signal with the default value of the terminal (all terminal's signals grow at the same rate).

Variables have retention properties. They differ from terminals in that if no assign activity is invoked during a computation step to extend the signal component of a variable then the system extends that signal with the present value. Variables are then much as found in programming languages.

Real Time Variables model abstract storage devices whose value may change only once per real time interval. When the transfer activity is invoked, the signal part of the real time variable is extended by one real time interval. Within an interval the computation step signal is extended with the first value - all step values are the same. A value is carried from interval when no transfer is invoked.

## 5. DESCRIPTION SEGMENTS

CONLAN is a formal semantic and syntactic base for the description of all phases in the design and documentation of digital systems. Previous sections have sketched the basic concepts and models underlying the whole CONLAN language family. For the rest of this paper we emphasize the logic system designer's point of view. By this, we mean the user of a CONLAN language rather than the definer of a member of the CONLAN language family.

DESCRIPTION segments provide the user with a structuring means to describe how a digital system is constructed from a set of components (network description). CONLAN does not go down to the electronic component level of a description; the most detailed description the user can write is therefore a logic gate network. The hardware designer however is not necessarily interested in gate-level details. He might not even know at some stage of a design, neither how many components he will be actually using, nor which specific ones, in some part of his system.

CONLAN provides maximum flexibility for stepwise refinement of a design. Instances of purely behavioral descriptions can be interconnected with instances of detailed network descriptions. Moreover, inside the same DESCRIPTION segment, some parts may be expressed in terms of clearly identified hardware elements where as other parts might be abstract.

CONLAN supports two ways of representing digital components. If the hardware designer is mainly interested in describing behaviour, he will define and invoke FUNCTION and ACTIVITY segments. If structure must be diplayed, he will define and instanciate DESCRIPTION segments. The choice is not dictated by the level of detail he wishes to display: all of these segments can be used for various levels of abstraction. Yet a fundamental difference exists as for how much hardware is implied in the designers text. If the same FUNCTION or ACTIVITY segment is invoked three times in a model, there is no a priori indication of the number of hardware units: there could be one (with multiplexing or time sharing) or as many as three. On the contrary, if a DESCRIPTION segment is used, there are exactly as many distinct hardware components as are instanciated. And, in fact, in a top-down approach, going from a model expressed in terms of operations to a model expressed in terms of instances of descriptions is precisely what is called synthesis.

## 6. CONCLUSIONS

To promode an orderly development of hardware description languages and to enhance their

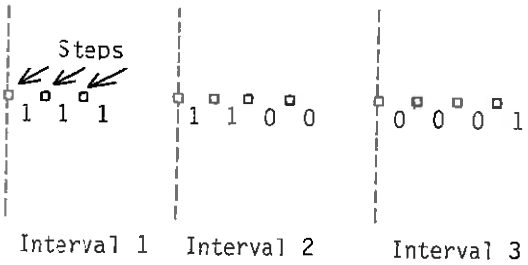
acceptance in an industrial environment, CONLAN provides a powerful construction mechanism for such languages, based on a common core syntax. This construction mechanism ensures that the semantics of the languages derived are well defined. Further, semantically related languages can be constructed which permit the description of digital systems at different levels of abstraction. The common core syntax facilitates learning a new language written in the CONLAN framework.

The concept of the interpreter is basic to the family of languages. The interpreter provides the basic counters used to model elapsed time (to and so ) and the detection of error conditions. It can also be augmented in a controlled manner by the toolmakers. This is achieved by the definition of special functions and activities which can then be invoked automatically by the interpreter.

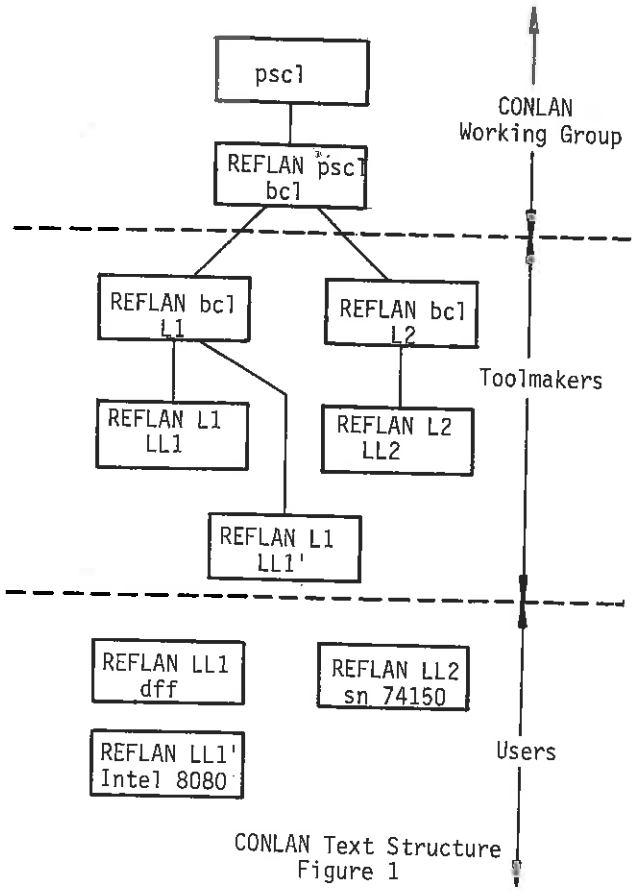
Base CONLAN is primarily a starting point, with well defined and semantically sound primitives, for language designers, to derive a coherent and comprehensive family of digital system description languages. The ability to partition a design into network of interconnected modules is supported by the notion of DESCRIPTION segments. Instanciation of such segments, and intercommunication between instances, directly depicts the structure of a system, and the data and control paths which connect its components.

ACKNOWLEDGEMENTS

The authors are indepted to Bell Northern Research (Ottawa), Sperry Univac (Philadelphia), Office of Naval Research, Ballistic Missile Defense Advanced Technical Center (Huntsville), IRIA (Paris), Bundesministerium für Forschung und Technologie (Bonn), Siemens (Munich) and Fujitsu (Tokyo) for their interest and support, Professor Yachan Chu for his early contributions, and in particular to Professor Jack Lipovski for his help and unwavering confidence in the group.




CONLAN Model of Time
Figure 2




CONLAN Text Structure
Figure 1

REFERENCES

[1] R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, F. Hill, P. Skelly: "CONLAN - A Formal Construction Method for Hardware Description Languages: Basic Principles". National Computer Conference, Volume 49, Anaheim, Cal. 1980.

[2] R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, F. Hill, P. Skelly: "CONLAN - A Formal Construction Method for Hardware Description Languages: Language Derivation". Proceedings National Computer Conference, Volume 49 Anaheim, Cal., 1980.

[3] R. Piloty, M. Barbacci, D. Borrione, D. Dietmeyer, F. Hill, P. Skelly: "CONLAN - A Formal Construction Method for Hardware Description Languages: Language Application". Proceedings National Computer Conference, Volume 49 Anaheim, Cal., 1980.

[4] M. R. Barbacci: "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems". IEEE Computer Society, Transactions on Computers, Volume C-24, Number 2, February 1975.

[5] Special issue on Hardware Description Languages, IEEE Computer Society, Computer, Vol. 7, No. 12, Dec. 1974

[6] Proceedings of the 2nd International Symposium on Computer Hardware Description Languages, Darmstadt, ACM German Chapter, Lectures-W, 1974.

[7] Proceedings of the 3rd International Symposium on Computer Hardware Description Languages and their Applications, New York, Sept. 3. - 5., 1975, IEEE Cat. No. 75 CH1010-8C.

[8] Proceedings of the 4th International Symposium on Computer Hardware Description Languages, Palo Alto, Oct. 8. - 9., 1979, IEEE Cat. No. 79 CH 1436-5C.

[9] Special issue on Hardware Description Languages, IEEE Computer Society, Computer, Vol. 10, No. 6, June 1977.

[10] Collection of Proceedings of the IEEE, ACM Design Automation Conference.

[11] Collection of Proceedings of the Fault Tolerant Computing Symposia.

[12] R. Piloty: "Guidelines for a Computer Hardware Description Consensus Language" (2nd draft), Memorandum to the Conference on Digital Hardware Languages, June 6., 1976.

[13] B. Liskov, S. Zilles: "Programming with Abstract Data Types", SIGPLAN Notices 9 pp 50 - 59, April 1974.

[14] W. A. Wulf, R. L. London, M. Shaw: "Abstraction and Verification in ALPHARD" Technical Report, Department of Computer Science, Carnegie-Mellon University, March 1976.

[15] P. Lucas, K. Walk: "On the Formal Description of PL/I", Annual Review of Automatic Programming, Vol. 6, part 3, 1969.